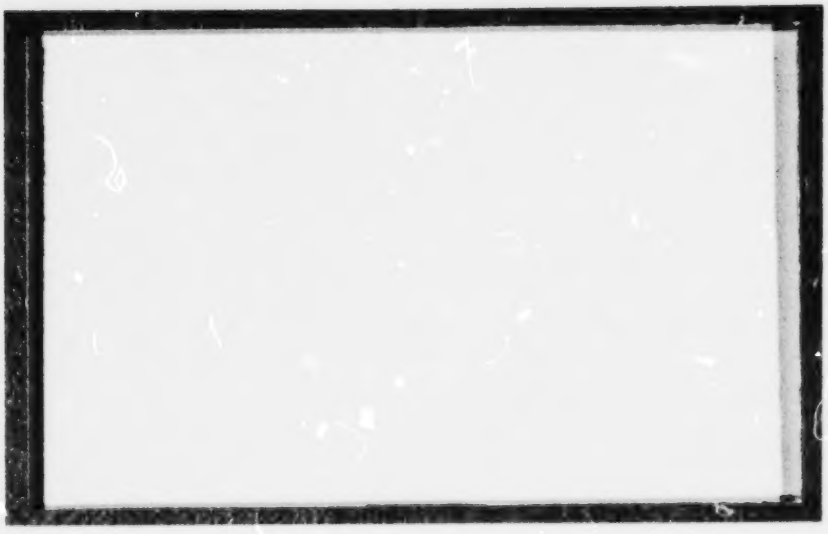
UNIVERSITÉ DE MONTRÉAL



CENTRE DE RECHERCHE SUR LES TRANSPORTS

CASE POSTALE 6128, SUCCURSALE A, MONTRÉAL, QUÉBEC, H3C 3J7

A DUAL SIMPLEX ALGORITHM FOR FINDING ALL SHORTEST PATHS

by

M. Florian, S. Nguyen and S. Pallottino*

* Consiglio Nazionale delle Ricerche, Istituto per le Applicazioni del Calcolo "M.Picone", Rome, Italy.

This research was supported by grants from the National Research Council of Canada and the National Research Council of Italy and by a grant from NATO System Sciences Panel.

Université de Montréal

Centre de recherche sur les transports - Publication # 139
Département d'informatique et de recherche opérationnelle - Publication # 325

Juin 1979 - Révisé en août 1980

Dépôt légal - 4e trimestre 1979

ABSTRACT

We present an adaptation of the dual simplex algorithm, for computing all shortest paths on a network. Given a shortest path arborescence rooted at node r, the change of root to a new origin s, renders the arborescence rooted at r dual feasible and primal infeasible for the new problem. The adaptation of the dual simplex algorithm to compute the shortest paths from node s results in an algorithm which has the flavor of a label setting method. It generally does not require the examination of all the nodes of the network. We report some computational results with the method which indicate that it is at least as efficient as successive applications of a label-setting or a label-correcting shortest path algorithm.

INTRODUCTION

The analysis of flows in transportation and communication networks is often carried out with the aid of steady state models of the flow during a given period of time, commonly known as traffic assignment models. These models are multi-commodity flow problems with convex [9,11], concave [14], or linear cost functions. Many of the algorithms used to solve these problems require the computation of shortest paths from all origin nodes to all destination nodes. Since the networks considered are relatively large and sparse, the computation of the shortest paths is achieved by considering each origin in turn and computing shortest paths to all destinations, by using either a label-correcting or a label-setting shortest path algorithm [5] (various ways of implementing the computations are presented in [2,3,8]). The use of matrix-multiplication type algorithms [6], which require the storage of the node/node incidence matrix, is impractical due to the size of the networks considered.

The consideration of each origin in turn often results in computing shortest path trees from nodes which are "close" to each other. Thus, in a large network, the resulting shortest path* trees share many arcs. This observation, although well known in the folklore, has not been previously exploited, mainly due to the remarkable simplicity and efficiency of the algorithms available to compute shortest paths from a given origin. All the information available from the computation of the shortest path tree from a given origin is usually discarded before the consideration of the next origin. It should be mentioned that Murchland [10] and Dionne [4]

---

* Due to the coloquial use in the literature we shall refer to paths, although we mean chains, that is a sequence of directed arcs; similarly we shall refer to trees although we mean arborescences.

studied the problem of finding the new shortest distances, given the matrix of shortest distances between all nodes of the network and a change in the length of an arc, in the context of matrix multiplication type algorithms. As mentioned before, the use of such methods is impractical.

The starting point for the work presented in this paper is the observation made by Gallo [7] in his recent work on the use of reoptimization procedures for the recomputation of shortest paths,when an origin change is made. The shortest path tree rooted at node r is an optimal solution to a corresponding linear program; when a next origin s is considered, the previous tree is a dual feasible and primal infeasible solution for the new problem. Gallo [7] suggested the use of the 'reduced costs' for the computation of the shortest paths from node s. The approach which we will explore is the adaptation of the dual simplex method to compute the shortest paths from the next node s.

The paper is organized in the following way: first, the formulation of the shortest path problem as a linear program and the dual simplex method are recalled; then, the dual simplex method is adapted for the reoptimization of the shortest path tree, and the properties of the resulting method are demonstrated; the design and the implementation of an efficient code are presented before finally reporting some computational results obtained with the code. The results strongly advocate the use of this reoptimization method for the shortest path calculations in the algorithms used to solve the multi-commodity flow problems outlined above.

THE SHORTEST PATH PROBLEM AND THE DUAL SIMPLEX METHOD

We consider a directed graph $G=(N,A)$, with node set $N=\{1,2,\ldots,n\}$ and arc set A, whose elements are ordered pair of nodes (i,j) indexed by a. Let $t(a)$ and $h(a)$ denote the tail and head of arc a and let c_a (c_{ij}) be the length associated with each arc.

$F(i) = \{a \mid t(a)=i,\ a\in A\}$ is the set of arcs that originate at node i and

$B(i) = \{a \mid h(a)=i,\ a\in A\}$ is the set of arcs that terminate at node i.

As is well known, the problem of finding the shortest paths from origin node r to all other nodes may be formulated as the linear program

$$\text{Min} \sum_a c_a \cdot x_a \tag{1}$$

subject to

$$\sum_{a\in F(i)} x_a - \sum_{a\in B(i)} x_a = b(i),\ i\in N \tag{2}$$

$$x_a \geq 0,\quad a\in A \tag{3}$$

$$\text{where } b(i) = \begin{cases} n-1, & i=r \\ -1, & i\neq r \end{cases} \tag{4}$$

Since no capacity restrictions are imposed on the arc flows each unit of flow from the origin r to the destination i will follow a shortest path from r to i in order to obtain the minimum of (1)

The corresponding dual linear program is, by letting $(-\pi_i)$ be the dual variable associated with constraint i,

$$\text{Max} \sum_j (\pi_j - \pi_r) \tag{5}$$

subject to

$$c_{ij} + \pi_i - \pi_j \geq 0,\ (i,j)\in A \tag{6}$$

A primal feasible solution for problem P (that defined by (1) - (4)) is a set of flows $\{x_a\}$ that satisfy (2), (3). A dual feasible solution for problem D (that defined by (5), (6)) is a set of dual variables $\{\pi_i\}$ that satisfy (6). An optimal solution is both primal and dual feasible.

It is well known [1] that every basis of (2) - (4) corresponds to the arcs of a spanning tree T of the network G and that an optimal solution is realized by a basic solution which minimizes (1). The flow values of the basic (in-tree) arcs are obtained as the unique solution of (2) - (4) obtained for a given T by setting $x_a = 0$, $a \notin T$. Also, (2) contains one redundant equation, thus one can set $\bar{\pi}_r = 0$ and then an optimal dual solution $\bar{\pi}_i$, $i \neq r$, gives the shortest distances from the origin to the node i.

The computation of the shortest paths from a next origin s amounts to changing the vector b(i) of problem P as follows:

$$\begin{aligned} b'(s) &= n - 1 \\ b'(r) &= -1 \\ b'(i) &= b(i), \quad i \neq r,s \end{aligned} \tag{7}$$

to obtain problem P' (defined by (1) - (3), (7)). The corresponding dual problem D' is given by

$$\text{Max} \sum_j (\pi_j - \pi_s) \tag{8}$$

and constraints (6).

If the optimal solutions of P and D, hence a tree T, are known, the basic arcs $a \in T$ are a basis for problem P' and the dual variables are feasible for problem D' as well. Since some of the flow values of the basic arcs

will not satisfy (3), one has a dual feasible but primal infeasible solution of P'. Thus it seems natural to consider the dual simplex method for solving P'.

In general terms the steps of this algorithm are as follows:

(0) start with a dual feasible, primal infeasible solution

(1) if the primal solution is feasible, terminate

(2) select the variable to exit the basis (usually it is the variable with the most negative solution value)

(3) select the variable to enter the basis such that dual feasibility is maintained

(4) obtain new basis, update primal and dual solution values and return to (1).

The adaptation of the dual simplex for solving P', which is described in the next section, takes advantage of the special structure of the shortest path problem. In particular, primal infeasibilities may be recognized from the orientation of the in-tree arcs with respect to the new root node s; $x_a < 0$ implies that arc a points toward the new root and the most negative valued basic variable may be determined without actually computing the solution of the basic variables. Thus, there is no need to update the primal solution values. Furthermore, the update of the dual variables may be done implicitly.

SOLVING P' BY THE DUAL SIMPLEX ALGORITHM

Consider problem P' and an initial basis tree T which is an optimal basis for problem P. T is an arborescence* rooted at node r. The new origin is now node s.

---

* We emphasize here the arborescence, although we have previously used tree instead.

The values of the flows of the basic arcs a, $a \in T$, may be characterized by the following analysis. Consider a node set $M \subseteq N$ of cardinality $|M|$ and its complement $\bar{M} = N - M$; let

$$b'(M) = \sum_{i \in M} b'(i), (M,\bar{M}) = \{a \mid t(a) \in M \text{ and } h(a) \in \bar{M}\} \text{ and } x(M,\bar{M}) = \sum_{a \in (M,\bar{M})} x_a.$$

By summing the conservation of flow equations for all $i \in M$, one obtains

$$b'(M) = x(M,\bar{M}) - x(\bar{M},M) \tag{9}$$

Note that, by (7), $b'(M) > 0$ if and only if $s \in M$.

Consider the (unique) simple path from s to a node i on T. Any arc in this path which is oriented towards s is called a *reverse* arc. Also let R(s,a) be the set of nodes i such that the path from s to i on T does not contain arc a.* It follows immediately from (9) that

Proposition 1: The values of the flows of the basic arcs $a \in T$ of P' are given by:

$$\bar{x}_a = \begin{cases} -b'(R(s,a)) & \text{if a is a reverse arc } (h(a) \in R(s,a)) \\ b'(R(s,a)) & \text{otherwise} \end{cases} \tag{10}$$

Thus the primal infeasibilities are all associated with basic reverse arcs. There are two corollaries that follow from the proposition which further characterize any dual feasible, primal infeasible solution of P'. Let T(i) denote the subarborescence of T rooted at node i and N(i) the node set of T(i). Then clearly

Corollary 1: All arcs $a \in T(s)$ are both primal and dual feasible.

---

* This notation is inspired from Cunningham's article entitled "A Network Simplex Method", Mathematical Programming 11, pp. 105-116 (1976).

Corollary 2: The values of the flows of the basic arc of P' are given by

$$\bar{x}_a = \begin{cases} |R(s,a)|-n & \text{if } h(a)\in R(s,u) \\ n-|R(s,a)| & \text{if } t(a)\in R(s,a) \end{cases} \qquad (11)$$

Demonstration: This follows from (7) since every node $\neq s$ has a demand of 1 unit, hence $b'(R(s,a)) = n - |R(s,a)|$.

Thus, it is not necessary to compute the flow values of the basic arcs in order to determine the primal infeasible arcs. Moreover, we can identify the most negative valued basic arc without computing the flow values by using

Proposition 2: For any primal infeasible basic solution, the basic arc with the most negative flow value is the first reverse arc on the path from s to r.

Demonstration: Let the simple path from s to r be formed of the arcs $\{e_1, e_2, \ldots, e_k, e_{k+1}, \ldots, e_m\}$ and let e_k be the first reverse arc. All arcs e_j, $j=k+1,\ldots,m$ are reverse arcs. Since $R(s,e_k)$ contains less nodes than $R(s,e_j)$, $j=k+1,\ldots,m$ and $b'(j)=-1$, $j\neq s$, it follows that $b'(R(s,e_k)) > b(R(s,e_j))$, $j=k+1,\ldots,m$ and hence $\bar{x}_{e_k} < \bar{x}_{e_j} < 0$, $j=k+1,\ldots,m$.

Initially, when the root switch is made from r to s, all arcs on the path from s to r are reverse arcs. The first arc of this path, say arc e, is chosen as the outgoing arc. In general, the first reverse arc e on the path from s to r is deleted from the basis tree T. This subdivides T into two arborescences, one of which is T(s); note that T(s) may be empty, in which case the subdivision splits the node s and T-{e}. The choice of an arc f to enter the basis tree must be made to relink the two arborescences. Since f must form a cycle with arc e it must have its head in

N(s) and tail in $\overline{N(s)}$(=N-N(s))or vice-versa. Since f must be chosen to maintain dual feasibility we have

Proposition 3: The incoming arc f is the arc that has the least reduced cost among all arcs from the arc set $(N(s),\overline{N(s)})$;

$$\bar{c}_f = \min_{a\in(N(s),\overline{N(s)})} \{c_a + \pi_{t(a)} - \pi_{h(a)}\} \qquad (12)$$

Demonstration: The pivot step is completed by entering f into the basis and updating the dual variables, so that the new reduced cost of f is zero. Suppose now that $f\in(\overline{N(s)}, N(s))$ enters the basis with $\bar{c}_f > 0$, since the solution is dual feasible. The resulting pivot step has the properties that dual feasibility is not maintained and the objective function decreases. The dual variables are updated by setting either $\pi_j \leftarrow \pi_j + \bar{c}_f$, $j\in N(s)$ or $\pi_j \leftarrow \pi_j - \bar{c}_f$, $j\in\overline{N(s)}$; in either case it is easily verified from (8) that the objective function decreases. Furthermore, the arc e which just left the basis is no longer dual feasible since, for instance, after the pivot $\pi_{h(e)} \leftarrow \pi_{h(e)} + \bar{c}_f$, which renders its new reduced cost to $-\bar{c}_f$. Thus f must be chosen among the arcs $a\in(N(s),\overline{N(s)})$ and, in order to maintain dual feasibility, it must have the least reduced cost.

The pivot operation is completed by entering f into the basis tree and by updating the dual variables. Replacing arc e by arc f in T increases T(s) by f and T(h(f)) since, by corollary 1, all arcs in T(h(f)) are primal feasible as well. The dual variables are updated prior to the updating of T(s).

The provisional form of the adapted dual simplex may be stated now as follows:

(0) Given the arborescence T rooted at node r, identify T(s) and N(s). Set $z = s$.

(1) Determine the basic reverse arc e with head z, if no such arc exists terminate.

(2) Find the entering arc f with the least reduced cost $\bar{c}_f$ from the candidate arc set $(N(s), \overline{N(s)})$.

(3)

3.1. Update the dual variables:

$$\pi_j \leftarrow \pi_j + \bar{c}_f \ , \quad j \in N(s)$$

3.2. Update the basis tree:

$$z \leftarrow h(f)$$

$$T \leftarrow T \cup \{f\} - \{e\}$$

$$T(s) \leftarrow T(s) \cup \{f\} \cup T(z)$$

$$N(s) \leftarrow N(s) \cup N(z)$$

Return to step 1.

We can prove now the following:

Proposition 4: The above algorithm terminates in at most (n-1) iterations.

Demonstration: At each iteration T(s) is augmented by at least one arc, and by corollary 1, no arc in T(s) will ever be deleted from the basis tree. Since a spanning arborescence contains (n-1) arcs, the result follows.

Corollary 3: A theoretical computational upper bound for the algorithm in this form is $O(n^3)$.

Demonstration: Suppose the algorithm requires (n-1) pivot steps. Then step (2) of the algorithm requires at most (n-1) evaluations of arcs that link N(s) to $\overline{N(s)}$ at the first iteration, 2(n-2) evaluations at the second iteration and in general k(n-k) evaluations at pivot step k. Thus in total it requires $\sum_{k=1}^{n-1} k(n-k)$ evaluations or $O(\frac{n^3}{6})$ additions and comparisons. The update of the dual variables requires $\sum_{k=1}^{n-1} (n-k)$ additions or $O(\frac{n^2}{2})$ operations.

In the following section we describe a refinement of the algorithm which does not require the update of the dual variables and replaces (12) with a more efficient procedure. These refinements reduce the computational upper bound to $O(n^2)$.

DESIGN AND IMPLEMENTATION OF A CODE FOR THE ALGORITHM

The most expensive part of the algorithm is the reevaluation of the reduced costs of the arcs $a \in (N(s), \overline{N(s)})$ in step (2) and the update of the dual variables in step (3.1). However, a close analysis leads to further important simplifications which obviate the explicit update of the dual variables at each iteration and the recomputation of the reduced costs of arcs that were considered at a previous iteration.

After a pivot step, let N'(s) be the node set of the updated subarborescence $T'(s) = T(s) \cup \{f\} \cup T(h(f))$, $z = h(f)$.

At the next iteration, the reduced costs of all arcs in the new candidate arc set $(N'(s), \overline{N'(s)})$ must be available for the determination of the next incoming arc. A straightforward method is to compute the reduced cost of all candidate arcs from the updated dual variables, as implied in the

above algorithm, which leads to the $O(n^3/6)$ computational complexity. Several observations permit the design of an efficient pivot step. First we note that the candidate arc set $(N'(s), \overline{N'(s)})$ may be partitionned into $(N(s), \overline{N'(s)})$ and $(N(z), \overline{N'(s)})$ and the reduced costs of the arcs in the first subset may be obtained directly by simply substracting the quandity $\bar{c}_f$ from their previous values. Only the reduced costs of the arcs in the subset $(N(z), \overline{N'(s)})$, which have not been computed prior to this iteration, must be computed from the dual variables. These computations may be achieved by using only the initial dual variable values, since both $N(z)$ and $\overline{N'(s)}$ are subsets of $\overline{N(s)}$ in the previous iterations. Furthermore, it is more efficient to leave the reduced costs of the arcs in the subset $(N(s), \overline{N'(s)})$ unchanged and *add* the quantity $\bar{c}_f$ to the reduced costs of the arcs from $(N(z),\overline{N'(s)})$, thus limiting all the computations to the subset $(N(z),\overline{N'(s)})$. Consequently, implementing the above scheme obviates the necessity of updating the dual variables at each iteration.

Next, we note that it is possible to reduce the cardinality of the candidate arc set $(N'(s),\overline{N'(s)})$, by keeping for each node $j \in \overline{N'(s)}$ only the candidate arc with the least reduced cost among all candidate arcs incident to node j. Thus, it is convenient to assign to each node j a label ℓ_j defined recursively by

$$\ell_j \leftarrow \ell_z \ , \qquad j \in N(z)$$

$$\ell_j \leftarrow \min_{i \in N(z)} \{\ell_j, c_{ij} + \pi_i - \pi_j + \ell_z\} \ , \ j \in \overline{N'(s)}$$

where the labels have been initialized at suitable high values for all nodes $j \neq s$, and the label of s has been set to zero.

The resulting steps (2) and (3) of the algorithm may now be restated as follows:

Step 2: Find the incoming arc f by determining

$$\ell_z = \min_{j \in \overline{N(s)}} \{\ell_j\}$$

f is the arc that realizes ℓ_z ; $z = h(f)$

Step 3: Update the node labels and the basis tree.

$$\ell_j \leftarrow \ell_z \ , \quad j \in N(z)$$

$$\ell_j \leftarrow \min_{i \in N(z)} \{\ell_j, c_{ij} + \pi_i - \pi_j + \ell_z\} \ , \ j \in \overline{N(s)} - N(z) \qquad (13)$$

$$T \leftarrow T - \{e\} \cup \{f\}$$

$$T(s) \leftarrow T(s) \cup \{f\} \cup T(z)$$

$$N(s) \leftarrow N(s) \cup N(z)$$

Return to step 1.

The correct values of the dual variables, that is those corresponding to $\pi_s = 0$, are computed at the termination of the algorithm by using only addition operations:

$$\bar{\pi}_j = \ell_j + \pi_j - \pi_s, \quad j \in N$$

In this form, the algorithm resembles a label-setting algorithm since labels are assigned to nodes, and by corollary 1, once an arc enters T(s) it is 'set' as part of the shortest path arborescence rooted at s.

However, if in step (3) the inclusion of T(z) on T(s) empties T-T(s), then one does not have to explore all the nodes in the network; in particular all nodes $i \in N(z)$ are not further explored. By contrast, if the network is connected, conventional label-setting or label-correcting algorithms must examine all nodes.

We can now characterize the computational complexity of the algorithm, as described by the last version of steps 2 and 3.

Proposition 5: The algorithm runs in time of at most $O(n^2)$.

Demonstration: Suppose the worst case which requires (n-1) pivot steps. At each pivot the recursion (13) is performed for all arcs with tail in N(z) and head in $\overline{N(s)}$-N(z). There are at most (n-1) such arcs at each pivot step. Step 2 determines the node with the minimum label and this requires at most (n-i) comparisons at iteration i. Thus the algorithm requires $O(n^2)$ operations.

The network data is stored in the standard "forward star" form in which all arcs $a \in F(i)$ for $i=1,2,\ldots,n$ are stored contiguously on a list. The basis tree T, separated into two subarborescences rooted respectively at r and s, is described by a predecessor function $P(\cdot)$ such that for every node $i \neq r$ and s, P(i) is the immediate predecessor of node i on the subarborescence. A dummy node (n+1) is added to be P(r) and P(s). The identification of nodes $i \in N(z)$ is achieved by using a preorder traversal function $TH(\cdot)$ and a last node function $LAST(\cdot)$. The nodes in N(z) are given by the sequence $\{z, TH(z), TH(TH(z)), TH(TH(TH(z))), \ldots, LAST(z)\}$. For a detailed description of these functions and the appropriate updating procedures see [3].

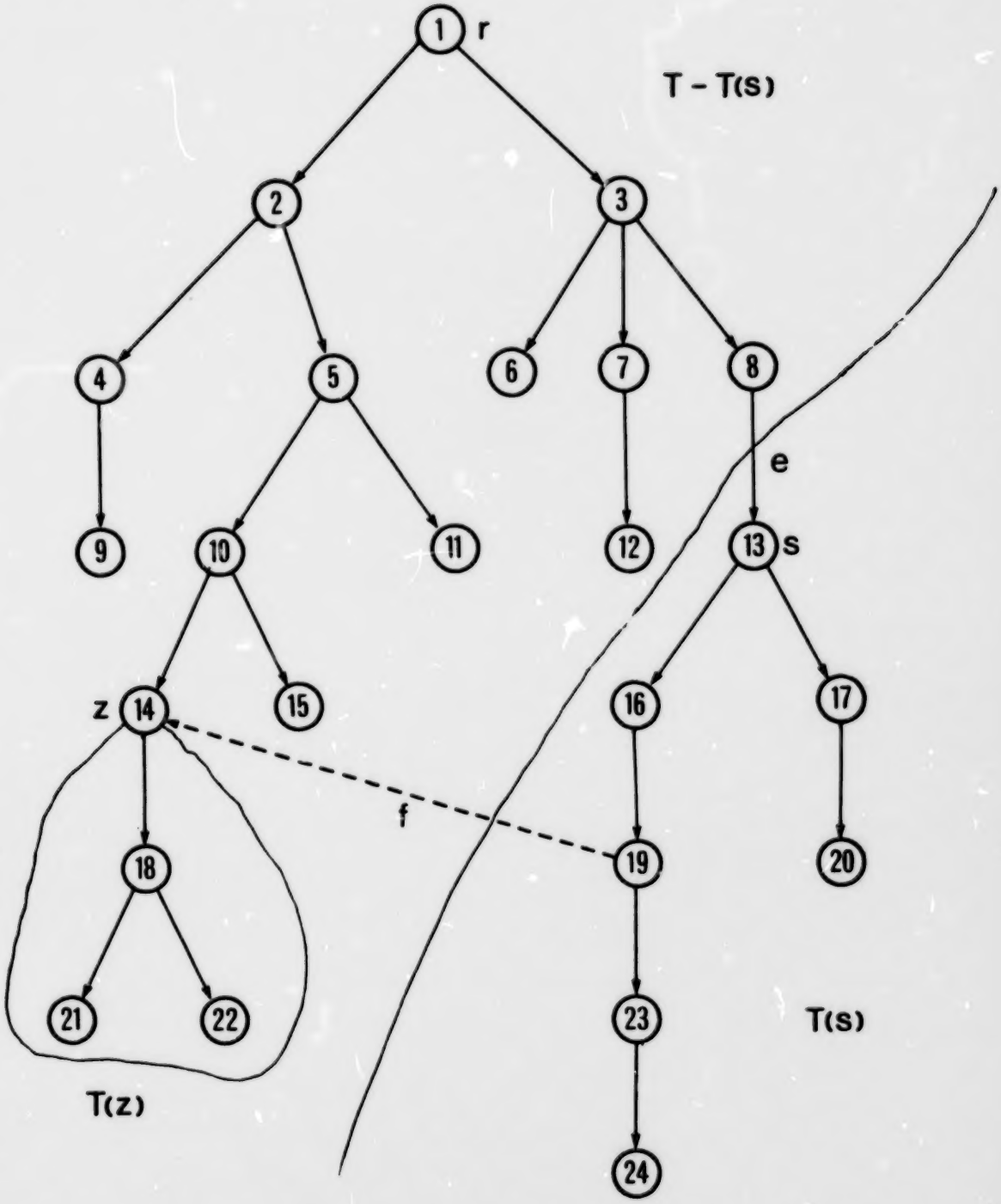


Figure 1: Illustration of root change, T(s) and a pivot, arc e is removed from basis, arc f enters. $T(s) \leftarrow T(s) \cup \{f\} \cup T(h(f))$

The node labels ℓ_j, $j \in \overline{N(s)}$ are stored in a binary heap with a 'dictionary' which keeps for each labeled node, its label and the head and the tail of the arc which is used to achieve the label. The first element of the heap provides ℓ_z, t(f) and z, and the heap is updated at every pivot step. The implementation of the algorithm with the binary heap requires $O(n^2 \ell nk)$ operations where k is the largest number of nodes with temporary labels. The n^2 term represents the number of arcs in a complete network. For sparse network the number of arcs |A| is much less than n^2 and the appropriate measure of computational complexity is $O(|A| \ell nk)$. Note that the update of the functions TH(·) and LAST(·) require operations of O(n) and $O(n^2)$ respectively.

COMPUTATIONAL RESULTS

The developed algorithm was coded in Fortran and tested on the CDC Cyber 173 of the Université de Montréal. The test networks represent the regional roads of the cities of Vancouver and Winnipeg. The network sizes are described in the following table. A shortest path tree must be computed from each centroid.

	Number of nodes	Number of arcs	Number of centroids
Vancouver	803	2181	150
Winnipeg	1040	2836	147

Table 1

Test networks sizes

Two sets of experiments were carried out. First, the shortest path trees rooted at the different centroids were computed by using first the arc lengths and then the arc (uncongested) travel times. The effects of using a "good" sequence of centroids were also evaluated. Then, we incorporated the algorithm in a traffic assignment procedure that is described in [12].

For all test runs, the first shortest path arborescence is produced by a label-correcting algorithm and then the dual algorithm is used.

The results that we obtained are summarized in tables 2 - 4.

*	Vancouver		Winnipeg			
			natural sequence		'good' sequence	
	lengths	travel times	lengths	travel times	lengths	travel times
number of pivots	75	78	132	141	73	85
number of nodes in the last subarborescence	90	68	80	67	105	99
total number of candidate arcs	155	163	253	262	165	180

* all values are average value over the total number of arborescences

Table 2

The average number of pivots required to find the new shortest path trees is remarkably low, considering that the worst-case bound is (n-1) or 802 for Vancouver and 1039 for Winnipeg. This clearly underlines the weakness of worst-case analysis. Another important characteristic of the algorithm is the size of the last subarborescence appended to T(s). Since

the nodes in this subarborescence are not explored, the larger the number of these nodes, the more efficient is the present algorithm over ordinary label-setting or label-correcting algorithms. For the networks tested, the average number of nodes in the last subarborescence is about 10% of the total number of nodes. Table 2 also shows the strong advantage of using a "good" sequence of centroids. A "good" sequence is a sequence of centroids in which successive centroids are relatively "close" on the network. This sequence was determined a priori.

In the first set of experiments, we used integer arc costs, the units are 1/1000th of a mile for the arc lengths and 1/100th of a minute for the arc travel times. Table 3 reports the running times obtained with the dual algorithm, the best label-setting algorithm for integer arc costs using the address calculation sort and Pape's [13] label-correcting algorithm (see Dial *et al.* [3], c.f. algorithms S1 and C2). The results indicate that in all the tests the dual algorithm was better than the other methods.

	Vancouver		Winnipeg			
			natural sequence		"good" sequence	
	lengths	travel times	lengths	travel times	lengths	travel times
Pape's label-correcting	25.0	20.8	55.5	39.9	55.5	39.9
Label-setting (address calculation)	30.9	38.4	33.4	37.7	33.4	37.7
Dual algorithm	20.2	20.7	28.4	29.9	24.4	25.1

Table 3 - All times in CPU seconds

	After 1st iteration	After the 11th iteration
Pape's label-correcting algorithm	42.5	362.1
Dual algorithm	25.1	268.9

Table 4 - All times in CPU seconds

Table 4 shows the computer running times for the shortest path calculations of a traffic assignment procedure using Pape's label-correcting algorithm (C2 in [3]) and the dual algorithm on the Winnipeg network. We recall that for each traffic assignment iteration a set of 147 shortest path trees must be computed. Arc travel times are real valued and change from a traffic assignment iteration to the next, hence we could not use the label-setting algorithm with the address calculation sort.

The improvements in the running times obtained suggest the use of the dual algorithm for the shortest path calculations in certain traffic assignment procedures, in spite of the fact that it requires two more node length vectors than the competing algorithms.

CONCLUSION

The algorithm described in this paper offers the possibility for making efficient use of previously developed shortest path information in computing shortest distance trees. The algorithm provided up to a 50% reduction in computer running time for a number of test problems when compared to two other efficient algorithms.

While this level of improvement may not in itself seem to be significant, this algorithm is one of many successive improvements offered over the past two decades in the computation of shortest paths. Aside from the computational considerations, the algorithm has a certain degree of intuitive appeal, because it efficiently uses previously generated information. The underlying idea of using components of other shortest path trees is so appealing, and the procedure relatively straightforward, it seems in retrospect quite surprising that, with all the efforts previously devoted to this subject, this approach has been left unexplored until now.

REFERENCES

[1] Dantzig, G.B., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey,(1963).

[2] Denardo, E.V. and Fox, B.L.,"Shortest Route Methods: 1. Reaching, Pruning and Buckets",Operations Research 27, pp. 161-186, (1979).

[3] Dial, B., Glover, F. Karney, D. and Klingman, D., "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees", Networks 9, pp. 215-248 (1979).

[4] Dionne, R., "Etude et extension d'un algorithme de Murchland", INFOR 16, pp. 132-146 (1978).

[5] Dreyfus, S.E., "An Appraisal of Some Shortest Path Algorithms", Operations Research 17, pp. 395-412,(1969).

[6] Floyd, R.W.,"Algorithm 97, Shortest Path", Comm. of A.C.M. 5, 345, (1962).

[7] Gallo, G.,"Reoptimization Procedures in Shortest Path Problems", Publication #132, Centre de recherche sur les transports, Université de Montréal, (1979).

[8] Gilsinn, J. and Witzgall, C., "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees", NBS Technical note 772, U.S. Department of Commerce, (1973).

[9] LeBlanc, L.J., Morlok, E.K. and Pierskalla, W.P., "An Efficient Approach to Solving the Road Network Equilibrium Traffic Assignment Problem", Transportation Research 9, pp. 309-318, (1975).

[10] Murchland, J.D. "A Fixed Matrix Method for All Shortest Distances in a Directed Graph and for the Inverse Problem", Ph.D. thesis, Karlsruhe University, (1970).

[11] Nguyen, S., "A Unified Approach to Equilibrium Methods for Traffic Assignment", Trarfic Equilibrium Methods, Lecture Notes in Economics and Mathematical Systems, 118, pp. 168-182, Springer-Verlag (1976).

[12] Nguyen, S. and James, L., "An Equilibrium Traffic Assignment Program", Publication #17, Centre de recherche sur les transports, Université de Montréal (1975).

[13] Pape, U., "Implementation and Efficiency of Moore's Algorithm for the Shortest Route Problem", Mathematical Programming 7, 212-222, (1974).

[14] Yaged, B.,"Minimum Cost Routing for Static Network Models" Networks 3, 315-331, (1971).